MAXIMUM AND MINIMUM SENSITIZABLE TIMING ANALYSIS USING DATA DEPENDENT DELAYS

A Thesis

by

KARANDEEP SINGH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2007

Major Subject: Computer Engineering

MAXIMUM AND MINIMUM SENSITIZABLE TIMING ANALYSIS USING DATA DEPENDENT DELAYS

A Thesis

by

KARANDEEP SINGH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Sunil P. Khatri
Committee Members,	Weiping Shi
	Hank Duncan Walker
Head of Department,	C. Georghiades

May 2007

Major Subject: Computer Engineering

ABSTRACT

Maximum and Minimum Sensitizable Timing Analysis Using Data Dependent Delays.

(May 2007)

Karandeep Singh, B.E., Panjab University-Chandigarh

Chair of Advisory Committee: Dr. Sunil P. Khatri

Modern digital designs require high performance and low cost. In this scenario, timing analysis is an essential step for each phase of the integrated circuit design cycle. To minimize the design turn-around time, the ability to correctly predict the timing behavior of the chip is extremely important. This has resulted in a demand for techniques to perform an accurate timing analysis.

A number of existing timing analysis approaches are available. Most of these are pessimistic in nature due because of some inherent inaccuracies in the modeling of the timing behavior of logic gates. Although some techniques use accurate gate delay models, they have only been used to calculate the longest sensitizable delay or the shortest topological path delay for the circuit. In this work, a procedure to find the shortest destabilizing delay, as well as the longest sensitizable delay of a static CMOS circuit is developed. This procedure is also able to determine the exact circuit path as well as the input vector transition for which the shortest destabilizing (or longest sensitizable) delay can be achieved.

Over a number of examples, on an average, the minimum destabilizing delay results in an improvement of 24% as compared to the minimum static timing analysis approach. The maximum sensitizable timing analysis results in an improvement of 7% over sensitizable timing analysis with pin-to-output delays. Therefore, the results show that the pessimism in timing analysis can be considerably decreased by using data dependent gate delays for maximum as well as minimum sensitizable timing analysis.

To my parents, sister, brother-in-law and my nephews

ACKNOWLEDGMENTS

I am very grateful to my advisor Dr. Sunil P. Khatri for giving me this opportunity to work under him. Without his constant guidance, suggestions and encouragement, this work would not have been possible. I owe him gratitude for showing me this way of research. He has supported and encouraged me whenever I needed him and answered all my questions very openly. The informal group meetings organized by him have been a constant source of knowledge and inspiration. I also want to thank him for all the facilities and support he has given to me. Thanks a lot Dr. Khatri for everything.

I would, also, like to express my sincere acknowledgment to Rajesh Garg, Nikhil Jayakumar and Kanupriya Gulati. Their constant support, valuable comments and guidance have helped me throughout the course of my master's study. They have also helped me learn new things, given me time to discuss problems, and have been a source of inspiration all along.

I would, also, like to thank my parents, sister and brother-in-law who taught me the value of hard work by their own example. I would like to share my moment of happiness with them. Without their encouragement and confidence in me, I would have never been able to pursue and complete my master's study.

Finally, I would like to thank all my friends who, directly and indirectly, supported and helped me in completing this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Timing analysis is a critical task in VLSI design today. *Static timing analysis* (STA) is the most commonly used type of timing analysis, since it is fast (linear in the size of the circuit). However, STA only identifies the *structurally* longest paths and does not consider the contribution of false paths. Thus, with the current thrust towards high performance devices, it is necessary to perform a more accurate analysis in estimating the maximum (or minimum) delay of our circuits.

There has been much research on *sensitizable timing analysis* (or *false-path aware* timing analysis) and on techniques to make this analysis more efficient [1, 2, 3, 4]. The objective of the maximum-delay sensitizable timing analysis is to determine the largest time when all the primary outputs of a digital circuit reach their stable final values, given the *maximum delays* of each gate in the circuit and the arrival times at the primary inputs of the circuit. While sensitizable timing analysis does perform a more accurate timing analysis than STA, it has still some inaccuracies which arise from the manner in which the delays of a gate are represented.

Another important timing analysis metric is the *shortest destabilizing delay* [5], which indicates the earliest time that the outputs become unstable after the inputs switch. This delay must be determine to check for *hold time violations* in a sequential circuit. Again, the existing approaches have handled this problem with an inaccurate gate delay model and hence have been pessimistic in their analysis.

In this thesis, sensitizable timing analysis is used to refer either to (i) maximum-delay sensitizable timing analysis or (ii) shortest destabilizing delay analysis, depending on con-

---

This thesis follows the style of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*

text. The main objective of this thesis is to present a technique to perform sensitizable timing analysis for static CMOS based combinational circuits while considering input arrival times and *data dependent gate delays*. Traditional timing analysis only utilizes the minimum (or maximum) pint-to-output delay of a gate to compute the shortest destabilizing (or longest sensitizable) path of the circuit. In static CMOS circuits, the differences in arrival times at the inputs of a gate can cause different delays at the output of the gate. This input transition dependence of gate delays is typically ignored in traditional timing analysis approaches. Our approach, on the other hand, utilizes the difference in arrival time and results in significantly more accurate sensitizable timing analysis numbers than regular STA and existing sensitizable timing analysis techniques. Although there exist some techniques which incorporate a data dependent delay analysis, they are used only to determine the longest sensitizable delay paths. In our technique, we are able to determine the longest sensitizable delay as well as the shortest destabilizing delay using a data dependent gate delay model. We use a unified formulation for both these problems with minor differences. In addition, we also determine the input vector transitions and the circuit path which sensitize both the longest sensitizable path and the shortest destabilizing path.

I-A. Previous Work

There has been a significant amount of work in the area of timing analysis. The first approach to compute the delay of the critical paths, without simulating all the input vectors, was based on the PERT (Program Evaluation and Review Technique) algorithm [6]. Since this approach determines the topologically longest (shortest) path without taking the logical dependencies into account, it is very likely to find a false path. The PERT delay was therefore used as an upper (lower) bound for the length of the critical paths. However, due to the growing complexity of combinational circuits the demand for tighter bounds gained

importance. This has resulted in different timing analysis approaches targeting the longest sensitizable (or shortest destabilizing) path of the circuit. In our approach we not only perform a sensitizable timing analysis, but also use a data dependent delay model to find the accurate maximum (minimum) delay of a circuit.

The initial path sensitization criteria are were based on the D-algorithm technique [7, 8]. Each of these criterion associates a given path with a set of logical conditions. The path is claimed to be *sensitizable* if there is no conflict in its set of conditions. A condition is denoted by a pair $\{lead,value\}$ which indicates that the signal *lead* has assumed an associated *value*. After deriving all the conditions for all the input leads to the gates along the path, the D-algorithm is used to propagate these original conditions to induce new conditions, and then to check whether there is a conflict between conditions. A conflict is flagged if any *lead* is required to stabilize at both logic 0 and logic 1 in the final set of conditions. The static sensitization criteria [7] detects a transition at the output of a gate if only the on-input of the gate is set to its controlling value and all the side-inputs are set to stable non-controlling values. However, it has been shown in [9] that the static criteria may overestimate or even underestimate the circuit delay. Hence it is not a reliable metric to find the critical delay of a circuit. Our approach is free from this problem since we do not intend to statically set the side-inputs of a gate as in [7] but compute the primary input vector transition which results the maximum (minimum) delay.

A dynamic sensitization criteria also proposed in [10, 11]. A path is dynamically sensitizable if and only if the side inputs of each node n are non-controlling at the arrival time of the sensitizing input signal at n. The dynamic sensitization criteria can result in an underestimate of the circuit delay when the *bounded delay model* [1] is used [10]. However, for fixed delays the dynamic sensitization is an exact criteria.

[1] The bounded delay model assumes that gate delays fall in a range [min,max].

Floating mode sensitization was proposed in [2]. This criterion was developed so as to satisfy the *monotone speedup property*[2] [10] under the bounded delay model. In this criterion, the state of each primary input is assumed stable but unknown before applying a change to a known value at time t . This assumption on the primary input behavior leads to waveforms on the internal nodes of the circuit which also have only one transition – from an unknown possibly changing value X to some final defined value. When such primary input behavior is assumed, the circuit is said to operate in *floating-mode*. Such a behavior yields a true upper bound to the critical path delay, but makes some conservative assumptions which may considerably overestimate the true critical delay. Our approach makes no such assumptions and is a closer bound to the true critical path delay. Also, since the floating-mode gate delay model does not specify the initial state of the primary inputs, it cannot be used to find the minimum sensitizable delay whereas our approach is more general and can be used to find both the maximum sensitizable delay and the minimum destabilizing delay.

In [3] the author explains that the false path problem inherently incorporates a delay model and a given solution is valid only in the context of the delay model considered. In order to compute the exact delay of a circuit the extended bounded delay-0 model (XBD0) model is introduced. Under this model, each gate has a maximum (positive) delay while the minimum delay is zero. The sensitization at each node is described in terms of a Boolean characteristic function which evaluates to the set of input vectors that sensitize a path. If the associated boolean expression of the path is computed to equal logic 0, the path is claimed to be a false path. This approach is further approximated in [4] by handling control and data paths separately. In both the approaches the problem of the dependence of critical path on the accuracy of the gate delay model persists. Also the XDB0 model used in this approach

[2]The monotone speedup property assumes that every circuit is infact a family of topologically identical circuits. For a true path on any member, there must exist a path of atleast the same length in the slowest member of the family.

assumes an unknown initial state, which cannot be used to find the minimum destabilizing delay. In our approach, we use a data dependent delay model which can be significantly less conservative than the XBD0 model, and is also equally effectively used in finding the minimum destabilizing delay, using a unified formulation.

In [12], Cheng et. al. introduce the concept of using the shortest destabilizing path to find the minimum sensitizable delay of a combinational circuit. A modified version of the *loose sensitization criteria* defined in [2] to find the shortest destabilizing path. It has been recognized in [13] that the destabilizing criteria used in [12] may lead to incorrect circuit clocking as it may lead to the destabilizing path having more delay than an exact sensitizable path. The author in [13], therefore introduces a *loose destabilizing criteria* [3] to correct this problem. In both [12, 13], the underlying gate delay model is still a pin-to-pin delay model which is inherently conservative. In our approach, we address the problem of finding the minimum destabilizing path under the data dependent delay model and achieve a true minimum destabilizing delay of the circuit.

The concept of timing analysis using data dependent delays was first introduced in [14]. A data-dependent delay model for analyzing the gate delay was developed. A modified topological longest-path algorithm is developed based on the data dependent delay model. The approach of [14] uses a path sensitization algorithm based on the *Loose Sensitization Criteria* [2] to generate the longest sensitizable path. This sensitization algorithm is based on a bounded delay model. Finally the two algorithms are combined – the authors iteratively generate the next longest path using the path sensitization algorithm and then use the topological longest-path algorithm to find the data dependent delay for the path generated. This is the first attempt to use data dependent delays, but the algorithm is used only to find

[3]The *loose destabilizing criteria* assumes a path to be destabilizing, if under a given input vector each input to a gate on the path is either controlling, or the earliest non-controlling input for its corresponding gate.

the longest data dependent sensitizable path. Our technique does a sensitizable timing analysis based on the data dependent delay model, Additionally, we are able to find the shortest destabilizing path without restoring to a two step procedure. Again, we are also able to determine the vector transitions that sensitize both the minimum and maximum delay paths which may have to be explicitly enumerated in [14]. Our formulation on the other hand lends itself to a fully implicit implementation.

In [15], Chen et. al. use a gate delay model for simultaneous input switching. This model uses a data dependent delay for simultaneous to-controlling transitions, whereas a pin-to-pin delay model is still used for simultaneous to-non-controlling transitions. The proposed model is used in STA and a tighter bound on the minimum delay is achieved. However, this technique may still be considerably conservative as it cannot be used to find the minimum destabilizing path. In our approach we use the data dependent delay model for *all* the transitions at the output of a gate (instead of only the simultaneous to-controlling transitions in [15]) and also use an accurate sensitization technique to be able to find the minimum destabilizing path.

I-B. Thesis Summary

The demand for high performance and the improvements in VLSI design techniques make it necessary for circuit analysis techniques to be as accurate as possible. Timing analysis is one of the most important step in evaluating the performance of a VLSI circuit and to determine the clock speed at which the circuit can operate. Traditional techniques like Static Timing Analysis are highly pessimistic in their approach. The existing sensitizable timing approaches have developed various sensitization criteria such as dynamic sensitization criteria, floating sensitization criteria and exact and loose sensitization criteria. These criteria are used to determine the longest and shortest sensitizable paths of a circuit. However, most

of these techniques do not incorporate accurate gate delay models which yield significantly more accurate timing estimates.

It has been recognized that using data dependency in the gate delay model can remove a great deal of pessimism in the earlier approaches. This idea of incorporating a more accurate estimate of the delay of a gate into a sensitizable timing analysis framework forms the basis of this thesis. Note that the proposed framework is generalized enough to be able to determine both the longest sensitizable and shortest destabilizing delays of a circuit. Our analysis also determines the circuit paths that result in the minimum and maximum delays, and the primary input vector transitions which sensitize these delays.

I-C. Thesis Organization

The rest of this thesis is organized as follows: Chapter II provide some background information which will be helpful in understanding the concept of sensitizable timing analysis. This chapter also highlights the inadequacies of the sensitizable timing analysis and details the data dependent gate delay model which is used for accurate timing analysis. Chapter III explains the approach of this thesis in performing "input arrival time aware" sensitizable timing analysis. In Chapter IV, experimental results are presented and in Chapter V the conclusions and future work are discussed.

CHAPTER II

BACKGROUND

In this chapter we discuss some of the basic concepts needed to understand the work in this thesis. First we discuss the setup and hold time violations. This is followed by a discussion on Static and Sensitizable Timing Analysis. Further, the data dependent gate delay model used in this thesis is explained, along with an analysis of the accuracy of this model.

II-A. Setup Time and Hold Time Violations

The setup time of a flip-flip F is defined as the minimum time before the arrival of the clock pulse by which the data at the input of the F must be stable. If the input to any flip-flip has an arrival time AT such that $AT > t_{clk} - t_{setup}$ then the *Setup Time Violation* is said to have occurred. Such a violation occurs in a sequential circuit when the sum of the maximum delay of the combinational circuit between two flip-flops and the setup time of the second flip-flop is more than one clock period.

Similarly, the hold time of the flip-flop F is defined as the minimum time that the input should remain stable at the input of F after the arrival of the clock edge, for it to be able to latch it properly. If the input to any flip-flip has an arrival time AT such that $AT < t_{hold}$ then the condition is called a *Hold Time Violation*. In a sequential circuit the *Hold Time Violation* usually occurs if the minimum delay of the combinational circuit between two flip-flips is smaller than the *hold time* of the second flip-flop.

Note that the *Setup Time Violation* depends on both the clock period and the maximum delay of the combination circuit can be corrected by running the circuit on a slower clock. However, the *Hold Time Violation* only depends on the minimum delay of the circuit. Hence, it is more critical to determine the minimum delay accurately. This work mainly concentrates on finding the accurate minimum delays of the circuit.

II-B. Static Timing Analysis (STA)

The objective of static timing analysis is to calculate the minimum and maximum delay at the primary outputs of any given digital circuit. It is a vectorless approach i.e. the timing analysis is done without using any input vectors. The maximum and minimum delay at the output pin for any given node n in the circuit are given by:

$$AT_n^{max} = \underset{n_i \ s.t. \ n_i \in FI(n)}{MAX} [AT_{n_i}^{max} + D_{n_i \to n}^{max}]$$
$$AT_n^{min} = \underset{n_i \ s.t. \ n_i \in FI(n)}{MIN} [AT_{n_i}^{min} + D_{n_i \to n}^{min}]$$

Here FI refers to the immediate fanins of the node n, AT_n^{max} and AT_n^{min} are the maximum and minimum arrival time at n and $D_{n_i \to n}^{max}$ and $D_{n_i \to n}^{min}$ and the maximum and minimum pin-to-output[1] delays of a gate from input pin n_i to the output n.

The minimum and maximum arrival times over all the primary output of the circuit define the lower and upper bound of the delay of the circuit respectively. However, the path followed to achieve these delays may be a false path (i.e. it may not be sensitizable) and these delays may never be achieved in the real circuit. Inspite of this deficiency, the major appeal of STA is its ability to provide a minimum and maximum delay estimate of the circuit in time that is linear in the size of the circuit.

II-C. Sensitizable Timing Analysis

The STA approach find an upper (lower) bound of the delay of the longest (shortest) topological path of the circuit. STA ignores the ignores the logical functionality of the circuit.

[1]The minimum(maximum) pin-to-output delays from the input n_i to the output n of a gate is defined as the minimum(maximum) delay when a transition at the input pin n_i produces a transition at output n.

Hence, the paths reported after STA may not even be sensitizable. When the functionality of the circuit is considered, there may be no possible assignment of values to the primary inputs which could cause a transition to propagate along this path. Such paths are known as *false paths*. They do not determine the delay of the circuit and should reported.

Techniques for timing analysis which implicitly or explicitly remove the false paths, and report the minimum (maximum) sensitizable paths are classified as *Sensitizable Timing Analysis* techniques. The generic approach in these cases is to develop a parameterized Boolean function called the *sensitization condition* or the *sensitization criteria*. The sensitization criteria determines if a transition at the primary inputs of a circuit can produce a transition at the primary outputs. The longest path of the circuit for which the sensitization criteria is met is called the *longest sensitizable path*. Similarly, the shortest path to meet the sensitization criteria criteria is called the *smallest destabilizing path*.

II-D. Gate Delay Model

In regular static timing analysis, we find the structurally worst case, (either minimum or maximum) circuit delay. In sensitizable timing analysis, false paths are implicitly removed from the analysis. In both these type of timing analysis, however, the method of propagating arrival times forward through a circuit are the same. They both consider the worst case delay of a gate when propagating arrival times. However, the delay of gate is not always the worst case value. It depends on the arrival times of the inputs to the gate. The difference in the results is explained below with a couple of examples. Let us first consider just the nominal delay of a NAND2 gate.

Table II.1 is a list of input transitions that cause the output of a 2-input NAND gate (with inputs a and b, and output c) to change its logic value. Let AT_i^{fall} denote the arrival time of a falling edge at signal i and AT_i^{rise} denote the arrival time of a rising edge at signal

Table II.1. Transitions for a NAND gate that cause its output to switch

Rising Transition #	ab → ab	Delay(ps)
1	11 → 00	30.5
2	11 → 01	50.5
3	11 → 10	53.0
Falling Transition #	**ab → ab**	**Delay(ps)**
1	00 → 11	55.3
2	01 → 11	46.5
3	10 → 11	42.7

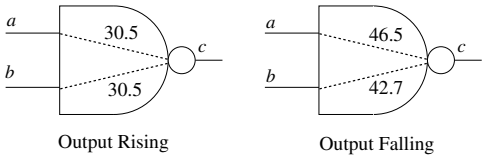


Fig. II.1. Pin to pin minimum delays for the NAND2 gate

i.

In the case of regular STA for calculating the minimum delay of a circuit (minSTA), the rising time (delay) at the output c of a NAND2 gate is calculated as

$$AT_c^{rise} = MIN[(AT_a^{fall} + MIN(D_{11\to00}, D_{11\to01})),$$
$$(AT_b^{fall} + MIN(D_{11\to00}, D_{11\to10}))]$$

where, $MIN(D_{11\to00}, D_{11\to01})$ is often referred to as the minimum pin-to-pin rising output delay from the input a, while $MIN(D_{11\to00}, D_{11\to10})$ is referred to as the minimum pin-to-pin rising output delay from the input b.

Similarly, in minSTA the falling time (delay) at the output c of a NAND2 gate is given by

$$AT_c^{fall} = MIN[(AT_a^{rise} + MIN(D_{00\to11}, D_{01\to11})),$$

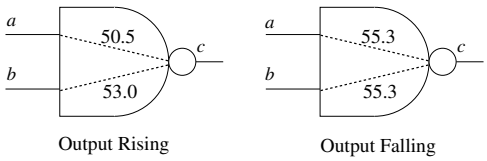


Fig. II.2. Pin to pin maximum delays for the NAND2 gate

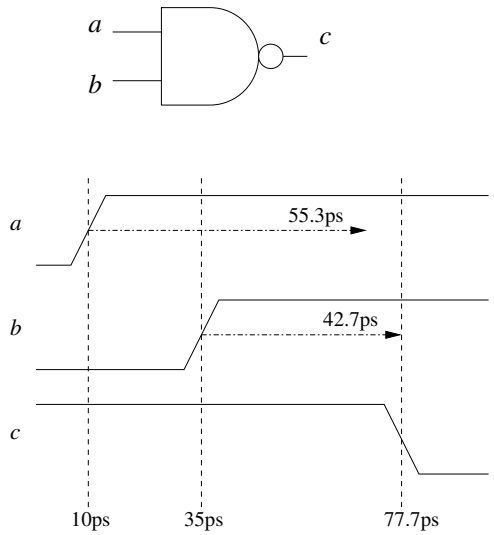
$$(AT_b^{rise} + MIN(D_{00\rightarrow 11}, D_{10\rightarrow 11}))]$$

where, $MIN(D_{00\rightarrow 11}, D_{01\rightarrow 11})$ is often referred to as the minimum pin-to-pin falling output delay from the input a, while $MIN(D_{00\rightarrow 11}, D_{10\rightarrow 11})$ is referred to as the minimum pin-to-pin falling output delay from input b. Figure II.1 illustrates the minimum pin-to-pin rising and falling delays and Figure II.2 illustrates the maximum pin-to-pin rising and falling delay for the example of Table II.1.

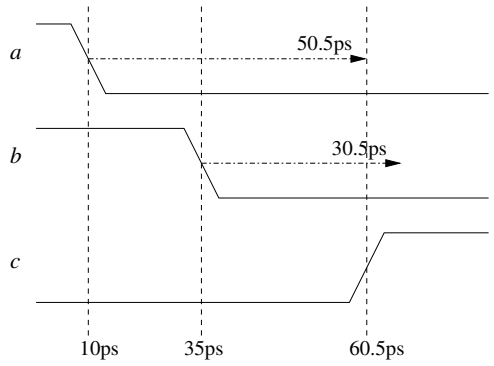
For example, if the falling or rising arrival time at inputs a and b was 10ps and 35ps respectively, then the rise delay at c would be calculated to be = MIN(10+30.5, 35+30.5) = 40.5ps. Similarly for a falling c output, the delay would be MIN(10+46.5, 35+42.7) = 56.5ps. However this is a pessimistic method of calculating the delay. In our approach we attempt to remove some of this pessimism.

Let us first consider the rising output. The output of the NAND2 gate switches high when any of the two inputs switches low. Such an input vector transition induces a transition on the gate output. Let us assume that this input transition was $11 \rightarrow 00$ for the NAND2 gate. Again assume that the input a and b arrive at 10ps and 35ps respectively.

Based on the arrival times, we can say that the gate effectively goes through the transition $11 \rightarrow 01 \rightarrow 00$ rather than $11 \rightarrow 00$ directly. Note that the output of the NAND2 gate



a) Falling Output



b) Rising Output

Fig. II.3. Example of Timing Analysis using a NAND2 gate

falls for the vector 01 as well. Hence, we calculate the delay to be

$$AT_c^{rise} = MIN((AT_a^{fall} + D_{11\rightarrow 01}), (AT_b^{fall} + D_{11\rightarrow 00}))$$

In our example, the delay is hence MIN(10+50.5,35+30.5) = 60.5. Figure II.3 (b) illustrates this graphically. Note that we used the minimum of the two delays in this case since any one input falling causes the output to switch. Also note that the rising delay calculated (60.5ps) is much larger than the minimum worst case rising delay calculated using minSTA (40.5ps). The reduction in pessimism in our approach occurs due to the fact that we have information about the input transition for the gate.

Now consider the case of the falling output. The output of the NAND2 gate switches low only when both the inputs switch high. Let us assume that the input transition for the NAND2 gate was $00 \rightarrow 11$. Additionally, we know that a arrives at 10ps and b arrives at 35ps. As a result, we can say that the gate effectively goes through the transition $00 \rightarrow 10 \rightarrow 11$ rather than $00 \rightarrow 11$ directly. Hence, in our approach, we calculate the delay to be

$$AT_c^{fall} = MAX((AT_a^{rise} + D_{00\rightarrow 11}), (AT_b^{rise} + D_{10\rightarrow 11}))$$

In our example, the delay is hence MAX(10+55.3,35+42.7) = 77.7. Figure II.3 (a) illustrates this graphically. Note that we used the maximum of two delays in this case since both inputs need to switch to cause the output to switch. Also note that the delay calculated (77.7ps) is again larger than the worst case minimum delay calculated using minSTA (56.5ps).

The accuracy of the computed falling and rising delays of our method were compared with SPICE [16]. The results are shown graphically in the Figures II.4 and II.5. These plots show the arrival time of the output c of a NAND2 gate, for the $00 \rightarrow 11$ and $11 \rightarrow 00$ transitions respectively. The arrival time of one of the inputs a is fixed to zero and the arrival time of the other input b swept between -150ps to 150ps. The output delays are

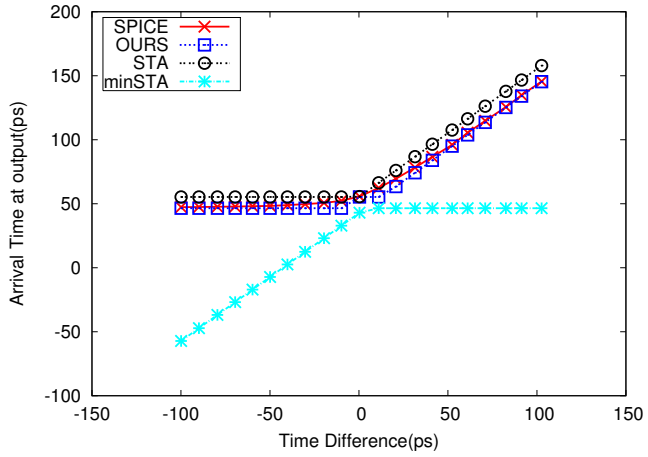


Fig. II.4. Plot of arrival times at output of NAND2 gate calculated through various means for the transition $00 \rightarrow 11$

shown for STA, minSTA and our method, along with the delay found by SPICE [16]. Figures II.6 and II.7 show the same analysis as Figures II.4 and II.5 respectively, but the sweep of the arrival times at *b* is restricted between -60ps and 60ps. Figures II.8 and II.9 show the relative error in the output delay value with respect to SPICE, with the arrival time of input *b* swept between -60ps and 60ps. As can be seen from these plots, our method of calculating the arrival times for multiple switching inputs matches SPICE quite accurately, (with an error of no more than 10% of the SPICE delay) and is significantly better than a traditional minSTA or STA method for computing arrival times (these methods have an error of upto 60 % as compared to SPICE delay).

We have thus seen how considering the data dependent delay of a gate (based on the input arrival times) can generate significantly different results than when considering just the minimum delay of a gate.

We now present an example to show how a sensitizable timing analysis (which uses the minimum delays of a gate) can give an inaccurate timing result. Consider the circuit in Figure II.10. Let the arrival times at the primary inputs be zero. The delays of the NAND2

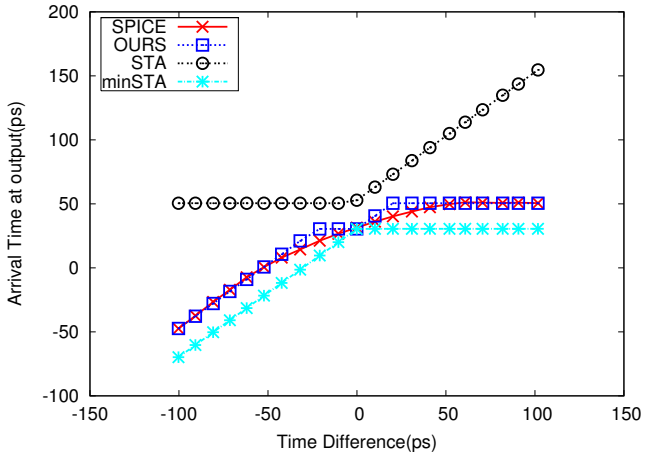


Fig. II.5. Plot of arrival times at output of NAND2 gate calculated through various means for the transition $11 \rightarrow 00$

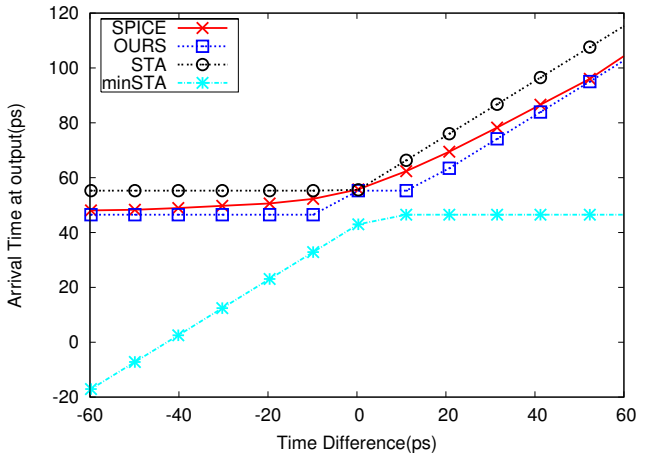


Fig. II.6. Same as the Figure II.4, with the input arrival time difference in the range of -60ps to 60ps

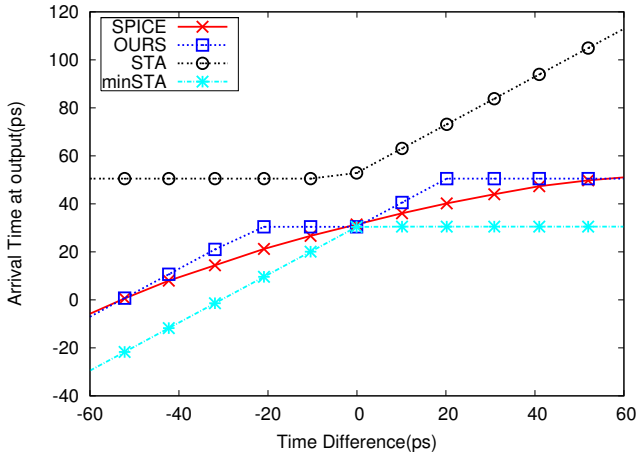


Fig. II.7. Same as the Figure II.5, with the input arrival time difference in the range of -60ps to 60ps

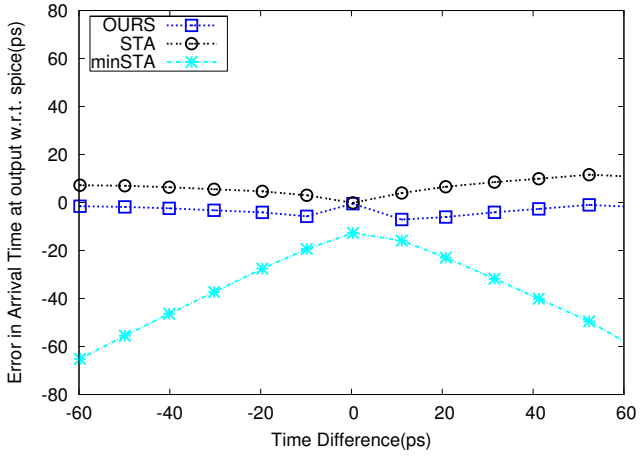


Fig. II.8. Plot of error in arrival times at output of NAND2 gate w.r.t. SPICE, for the transition 00 $\rightarrow$ 11

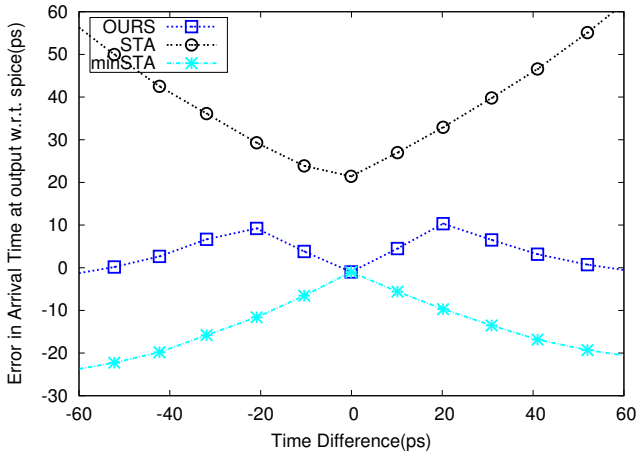


Fig. II.9. Plot of error in arrival arrival times at output of NAND2 gate w.r.t. SPICE, for the transition $11 \rightarrow 00$

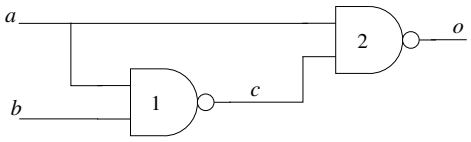


Fig. II.10. Example of a circuit where Sensitizable Timing Analysis can be inaccurate

gate used are given in Table II.1. From this table we can state that the pin-dependent minimum rising delay for the output of the NAND2 gate is 30.5ps for both the input pins. Similarly the pin-dependent falling delay for the output of the NAND2 labeled '2' is 46.5ps for the input pin a and 42.7 for the input pin b. This is shown graphically in Figure II.1. Considering the pin-to-pin delay model the shortest delay for the circuit in Figure II.10 is 30.5ps. This delay is sensitizable when the internal node c remains is non-controlling (i.e. $c = 1$). One of the sensitizable vector transitions (on the primary inputs) is 10→00 which causes the node c to remain at a stable 1 and the output node o to switch high at 30.5ps under the pin-to-pin delay model (since the minimum delay for the output rising is 30.5ps

from either input). Hence the output rising delay of 30.5ps is the minimum destabilizing delay using the pin-to-pin delay model.

However, when we utilize the data dependent delay model (from Table II.1), we get a different delay value. Let us first validate the minimum delay transition generated by the pin-to-pin delay model, using the data dependent delay model. Let the initial state of the output be a stable 0. When the primary inputs transition is 10→00, the gate labeled 1 (which drives node c) does not change the state of the output and it remains at stable 1 from time t =0. So the input transition on the gate labeled 2 (which drives the primary output o) is 11→01. From Table II.1 the delay for this transition is found to be 50.5ps. This delay clearly more than that obtained using a pin-to-pin delay model (which reported a delay of 30.5ps). Now let us consider the initial state of the output being stable 1. We do this to check whether the minimum falling delay under the data dependent delay model yields a value smaller than 50.5ps. The minimum transition delay can be achieved when the internal node c is non-controlling. When the transition on the primary inputs of the circuit is 00→10 the node c remains at a stable 1 state. So the transition on the inputs of the gate labeled 2 (which drives the output node o) is 01→11. The delay for this transition according to Table II.1 is 46.5. This is therefore the minimum destabilizing delay of the circuit under the data dependent delay model. From the above example it is clear that the pin-to-pin delay model not only is underestimates the smallest destabilizing delay but also can generate a wrong input vector to sensitize the delay. Hence, the data-dependent delay model is a closer estimate to the true behavior of the circuit.

Now, let us see how we incorporate the data-dependent delay model in our sensitizable timing analysis approach. We represent the feasibility of a transition at any node n at a time t in terms of its transition function τ. The transition function $\tau^t_{n,s}$ of a node n in the circuit is defined as the the set of transitions at the primary inputs for which the node n has a state $s \in \{0, 1, rise, fall\}$ (where 0 and 1 represent a corresponding stable state at n and *rise* and

$fall$ represent a transition at n) at time t.

Again consider the example of a NAND2 gate. Let the delay corresponding to each transition be represented as $D_{init\_vec \rightarrow final\_vec}$, where $init\_vec$ and $final\_vec$ are the initial and final input vectors at the inputs of the node. The output c can rise if any one of the inputs is at a stable 1 and the other input falls, or if both inputs fall at the same time.

Hence the transition function representing a rising transition at node c is given by:

$$\begin{aligned}
\tau^{t}_{c,rise} = {} & \tau^{t-D_{11\rightarrow 10}}_{a,1} \cdot \tau^{t-D_{11\rightarrow 10}}_{b,fall} \\
& + \tau^{t-D_{11\rightarrow 01}}_{a,fall} \cdot \tau^{t-D_{11\rightarrow 01}}_{b,1} \\
& + \tau^{t-D_{11\rightarrow 00}}_{a,fall} \cdot \tau^{t-D_{11\rightarrow 00}}_{b,fall} \\
& + \Sigma\tau^{(t-D_{11\rightarrow 01})<t_i<(t-D_{11\rightarrow 00})}_{a,fall} \cdot \tau^{t-D_{11\rightarrow 00}}_{b,fall} \\
& + \tau^{t-D_{11\rightarrow 00}}_{a,fall} \cdot \Sigma\tau^{(t-D_{11\rightarrow 10})<t_i<(t-D_{11\rightarrow 00})}_{b,fall}
\end{aligned} \tag{2.1}$$

We can do a similar analysis for the falling transition at the output node c for the same gate. The output c falls if one of the inputs is at a stable 1 and the other input rises, or if both the inputs rise at the same time. The transition function representing a fall transition at node c is given by:

$$\begin{aligned}
\tau^{t}_{c,fall} = {} & \tau^{t-D_{00\rightarrow 11}}_{a,1} \cdot \tau^{t-D_{10\rightarrow 11}}_{b,rise} \\
& + \tau^{t-D_{01\rightarrow 11}}_{a,rise} \cdot \tau^{t-D_{00\rightarrow 11}}_{b,1} \\
& + \tau^{t-D_{00\rightarrow 11}}_{a,rise} \cdot \tau^{t-D_{00\rightarrow 11}}_{b,rise} \\
& + \Sigma\tau^{(t-D_{00\rightarrow 11})<t_i<(t-D_{01\rightarrow 11})}_{a,rise} \cdot \tau^{t-D_{00\rightarrow 11}}_{b,rise} \\
& + \tau^{t-D_{00\rightarrow 11}}_{a,rise} \cdot \Sigma\tau^{(t-D_{00\rightarrow 11})<t_i<(t-D_{10\rightarrow 11})}_{b,rise}
\end{aligned} \tag{2.2}$$

Each row in the above expressions represent a separate condition that makes the output

node c sensitizable at time t, depending on the transition functions of its inputs a and b. Each row of the above expression will be referred to as a *term* of the transition function. Note that each *term* may require the evaluation of the transition function a fanin of c at a particular time instant or at a range of times. Each *term* represents a different condition that triggers a transition (rising /falling) at node c.

Also, note here that the stable 1(stable 0) condition for the input pins (a,b) is denoted by a single time instant (lets say t') for the sake of simplicity of representation. It actually represents the range of time from t' to t, when the transition occurs at the output node c at time t. For example, the term $\tau_{a,1}^{t-D_{11\rightarrow 10}}$ in the first expression for the transition function $\tau_{c,rise}^{t}$ denotes that the input a remains at a stable 1 from time $t - D_{11\rightarrow 10}$, upto time t.

We will use a recursive formulation to find the transition function at each node until the primary inputs are reached. At any primary input x a transition can occur only at $arr(x)$, which is a user specified arrival time of the primary input x. The transition function of a primary input x is expressed in terms of its *input state* at the time t. We define the *input state* x_s of a primary input x as a four-valued variable which, for $s \in \{0, 1, rise, fall\}$, indicates that node n is either

1. Statically 0: This means that x was statically 0 from time t = -∞, and never transitioned (i.e. arr(x) = -∞).

2. Statically 1: Similarly, this means that x was statically 1 from time t = -∞, and never transitioned (i.e. arr(x) = -∞).

3. Rising: This means that x rose at an infinitesimally small delay after arr(x).

4. Falling: Similarly, this means that x fell at an infinitesimally small delay after arr(x).

At the primary inputs(x), the transition function can be defined as

$$\tau^t_{x,rise} = \begin{cases} x_{rise} & \text{if } t = arr(x) \\ 0 & \text{otherwise} \end{cases}$$

$$\tau^t_{x,fall} = \begin{cases} x_{fall} & \text{if } t = arr(x) \\ 0 & \text{otherwise} \end{cases}$$

$$\tau^t_{x,1} = \begin{cases} x_1 & \text{if } t \leq arr(x) \\ x_{rise} + x_1 & \text{if } t > arr(x) \end{cases}$$

$$\tau^t_{x,0} = \begin{cases} x_0 & \text{if } t \leq arr(x) \\ x_{fall} + x_0 & \text{if } t > arr(x) \end{cases} \tag{2.3}$$

Note that the first condition in $\tau^t_{x,1}$ and $\tau^t_{x,0}$ is reasonable since x rises (falls) an infinitesimal delay after arr(x). With this ability to model $\tau^t_{x,s}$ (where $s \in \{0, 1, rise, fall\}$) in terms of arr($x$) our framework can elegantly incorporate the situation where primary inputs arrive at arbitrary times.

We now explain the need for the condition that an input rises or falls an infinitesimal delay after arr(x). Consider the NAND2 gate of Table II.1. Suppose we want to know if the output can fall at 42.7ps. This is done by a call to $\tau^{42.7}_{c,fall}$. When this call is expanded, one of the terms is:

$$\tau^0_{a,1} \cdot \tau^0_{b,rise}$$

Note that if arr(a) = arr(b) = 0ps, $\tau^0_{a,1} = a_1$ and $\tau^0_{b,rise} = b_{rise}$. Based on the delays of Table II.1, the stable 1 condition on a and the rising condition on b does yield a delay of 42.7ps.

Now, if we did not assume that an input rises slightly after arr(x), then we would have

$\tau^0_{a,1} = a_1 + a_{rise}$. Note that $\tau^0_{b,rise}$ still evaluates to b_{rise}. These transitions for the primary inputs a and b suggest that a delay of 42.7ps on c is possible in two ways – (i) a and b both rise at t = 0ps and (ii) a is stable 1 and b rises at t = 0ps. While the later transition function correctly yields a delay of 42.7ps as per Table II.1, the former actually yields a delay of 55.3ps. Hence the former transition is incorrect. If an input x is assumed to rise slightly after arr(x), then this error is avoided.

II-E. Conclusion

This chapter briefly covered the background information which will be required to understand this thesis. The next chapter will describe the algorithm for the data dependent sensitizable timing analysis.

CHAPTER III

DATA DEPENDENT SENSITIZABLE TIMING ANALYSIS

III-A. Introduction

Sensitizable timing analysis (as described in Chapter I) is used to perform false-path aware timing analysis. There exist various sensitizable timing analysis approaches, all of which take into account the logical functionality of the circuit. This helps them to correct the pessimism of STA, by removing the topologically long (short) paths which can never be sensitized due to the functionality of the circuit. However, the underlying gate delay model used by these approaches (the pin-to-output delay model) does not allow them to accurately model the behavior of a circuit. As seen in Chapter II, we can say that the true delay of the circuit can be significantly different from the delay estimated by any technique using a pin-to-output gate delay model. This thesis formulates an approach to use a data dependent delay model for performing sensitizable timing analysis. This approach can be used to find the *both* the minimum destabilizing and the maximum sensitizable delay of the circuit under the improved gate delay model.

This chapter presents the key contribution of this thesis i.e. a sensitizable timing analysis framework to evaluate the minimum destabilizing and the maximum sensitizable delay of a digital circuit using a data dependent gate delay model. In the next section, the overall approach is demonstrated with the example circuit discussed in Section II-D. Sections III-C and III-D formalize the methodology, and presents the main algorithms of the approach. The Section III-F discusses the implementation details, along with the pruning techniques used to make the procedure more efficient.

III-B. Example

Let us revisit the example of the circuit in Figure II.10. The delays for the NAND2 gate are given in Table II.1. This example demonstrates how the timing analysis approach proposed in this thesis is able to perform a sensitizable timing analysis utilizing the data dependent gate delay model.

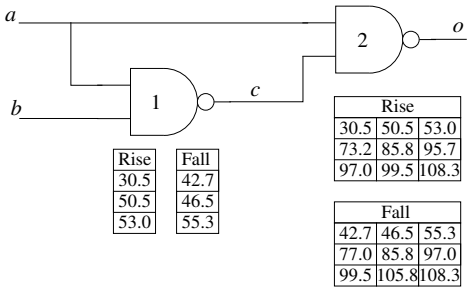


Fig. III.1. Example of a circuit accurately solved using data dependent delay model

As a first step, we obtain a list of all the possible rise/fall times at the each node of the circuit. These lists are illustrated in Figure III.1. The lists are generated by a topological traversal from the primary inputs to the primary outputs of the circuit. The list of all the possible rise/fall times at the output of a gate is calculated using the corresponding lists at its inputs and its data dependent delays. At this point, only the rise/fall times have been calculated but no check for sensitizability has been performed.

We also generate the list of input values for which each node remains at a static 0 or static 1 value from time $t = 0$ ps. These lists are calculated at each node n in the circuit and are called the *stable0 / stable1* lists for the node n. Each element of these lists is a cube in the offset (onset) of the logic function of node n. Note that the initial value at any primary input x is assumed to be stable at time t = -∞ and the only possible transition is at t = arr(x). This assumption allows us to use the *stable0 / stable1* lists to evaluate the logical conditions under which each node n is a static 0 / static 1 at a time earlier than its arrival

time. For the sake of simplicity let us assume that the arrival times of all the inputs of the circuit are 0 ps.

In order to check for the minimum destabilizing path we evaluate the worst case (minimum) delay among all the primary outputs and check for its sensitizability. In this example we have only one primary output, so we start from its lowest possible delay (30.5 ps – from the "Rise" list for node o in Figure III.1. The transition function for the output o to rise at time 30.5 ps is given by:

$$\begin{aligned}
\tau_{o,rise}^{30.5} &= \tau_{a,1}^{30.5-53.0} \cdot \tau_{c,fall}^{30.5-53.0} \\
&+ \tau_{a,fall}^{30.5-50.5} \cdot \tau_{c,1}^{30.5-50.5} \\
&+ \tau_{a,fall}^{30.5-30.5} \cdot \tau_{c,fall}^{30.5-30.5} \\
&+ \Sigma\tau_{a,fall}^{30.5-50.5<t_i<30.5-30.5} \cdot \tau_{c,fall}^{30.5-30.5} \\
&+ \tau_{a,fall}^{30.5-30.5} \cdot \Sigma\tau_{c,fall}^{30.5-53.0<t_i<30.5-30.5}
\end{aligned}$$

Let us analyze each term of this expression. In the first term the transition function $\tau_{a,1}^{30.5-53.0}$ requires the primary input a to remain static 1 from t = -22.5 ps to t = 30.5ps. This would be given by the *stable1* list at the node a. However, the fall time for c is required to be -22.5 ps which is clearly not feasible. A transition is said to be infeasible if it yields a transition function equal to 0. Thus, first term does not result in a feasible transition at the output c. Similarly, for the second term the static 1 on node c will be given by its *stable1* list (since t = -22.5 ps) but the falling transition at a at t = -22.5 ps is infeasible. Clearly, the transition at a is not feasible. In the third term, the transition function $\tau_{a,fall}^{30.5-30.5}$ evaluates to a_{fall} but the transition function $\tau_{c,fall}^{30.5-30.5}$ is again infeasible. The falling transition at node c is infeasible simply because the time t = 0 ps does not occur in the *fall time list* for node c (see Figure III.1. In the fourth term, the range of falling transitions at a is t =

(30.5 - 50.5) ps to (30.5 - 30.5)ps. Note that the inequality calculates the possible transition functions for a falling between -20 ps and 0 ps. This is an infeasible range for a since the primary inputs are allowed only to transition **at** 0ps. Again, in the fifth term for the node c the range t = (30.5 - 53.0) to (30.5 - 30.5) does not contain any valid value from the *fall time list* of the node c. Since no term results in a feasible transition function, we say that the rising output transition at 30.5 ps is not sensitizable.

Since the output fails to transition at t = 30.5ps, we evaluate the next lowest output transition from the Rise and Fall lists for node o. Therefore we next test if a falling edge at o at time t = 42.7ps is feasible. The transition function for o to fall at t = 42.7 ps is given by:

$$\begin{aligned}\tau_{o,fall}^{42.7} = \tau_{a,1}^{42.7-55.3} \cdot \tau_{c,rise}^{42.7-42.7} \\ + \tau_{a,rise}^{42.7-46.5} \cdot \tau_{c,1}^{42.7-55.3} \\ + \tau_{a,rise}^{42.7-55.3} \cdot \tau_{c,rise}^{42.7-55.3} \\ + \Sigma\tau_{a,rise}^{42.7-55.3<t_i<42.7-46.5} \cdot \tau_{c,rise}^{42.7-55.3} \\ + \tau_{a,rise}^{42.7-55.3} \cdot \Sigma\tau_{c,rise}^{42.7-55.3<t_i<42.7-42.7}\end{aligned}$$

By doing a similar analysis as in the first case, we can check that none of the terms of the above transition function results in a feasible transition at the output o. So the sensitization of the falling output transition at t = 42.7 ps fails.

Now we consider the next higher possible transition time at o, which is 46.5ps. The possible output transition at this time is falling. The transition function for o to fall at t = 46.5 ps is given by:

$$\tau_{o,fall}^{46.5} = \tau_{a,1}^{46.5-55.3} \cdot \tau_{c,rise}^{46.5-42.7}$$

$$+\tau_{a,rise}^{46.5-46.5} \cdot \tau_{c,1}^{46.5-55.3}$$

$$+\tau_{a,rise}^{46.5-55.3} \cdot \tau_{c,rise}^{46.5-55.3}$$

$$+\Sigma\tau_{a,rise}^{46.5-55.3<t_i<46.5-46.5} \cdot \tau_{c,rise}^{46.5-55.3}$$

$$+\tau_{a,rise}^{46.5-55.3} \cdot \Sigma\tau_{c,rise}^{46.5-55.3<t_i<46.5-42.7}$$

Only the second term of interest here. All of the others can be removed with similar analysis as was done for the first 2 cases. The condition for the second term is $\tau_{a,rise}^{0}$ (a rising transition on *a at t* = 0 ps), which evaluate to a_{rise}, and $\tau_{c,1}^{-8.8}$ (*c* is a static 1 from *t* = -8.8 ps to 46.5 ps). The transition function $\tau_{c,1}^{46.5-55.3}$ evaluates to the node *c* rising at a time before t_1 = -8.8 ps and not falling till t_2 = 46.5 ps. Since $t_1 < 0$ ps, the valid input states are calculated using the *stable1* list for *c*. This is done because of the initial assumption that a static 0 (static 1) value at a primary input is treated as though the input has fallen (risen) at time t = -∞. Thus, conditions generated for $\tau_{c,1}^{46.5-55.3}$ is a_0 + b_0. Using the above two expressions the final transition function at the output is given by:

$$\begin{aligned}\tau_{o,fall}^{46.5} &= \tau_{a,rise}^{46.5-46.5} \cdot \tau_{c,1}^{46.5-55.3} \\ &= (a_{rise}) \cdot (a_0 + b_0) \\ &= a_{rise} \cdot b_0\end{aligned}$$

This example illustrates how the approach followed in this algorithm not only generates the true minimum destabilizing delay, but also finds the correct input vector transition which sensitizes this delay. Note that the same formulation can find the maximum sensitizable delay of the circuit as well.

III-C. Data Dependent Shortest Destabilizing Delay Algorithm

This section presents the main algorithm to perform data dependent sensitizable timing analysis. Consider a Boolean network η. First the network η is decomposed and mapped using 2-input NAND gates and inverters only. Let the modified network be represented by η^*. Now η^* is sorted in a breadth-first manner. The resulting array of nodes is sorted in *levelization*[1] order, and placed into an array *L*. Thus the nodes of η^* are stored in *A* in topological order from the inputs to the outputs.

Now a node *n* is fetched from the array *A* in index order. Then the **generate_transition_lists** routine is used to generate the rise/ fall time lists at the node *n* using the rise/fall time lists of its immediate fanins and a data dependent gate delay model. Since sensitization is not checked in the creation of these lists, the values present in the list may correspond to the delay values which are not sensitizable by any primary input vector transition. After this, the **generate_stable_condition** generates the *stable0/ stable1* lists for the node *n*. As discussed in the previous example, these lists contain the input vectors which would make the node a static 0 or static 1 respectively.

The **next_shortest_delay** routine starts by selecting the shortest (rising or falling) delay at any primary output of the circuit. It returns the shortest delay t_d, the primary output node *o* at which the delay t_d may be achieved, and the output state *s* corresponding to the delay t_d. Successive calls to this routine return the next higher value of the delay, along with the corresponding node and transition values. Note that although $s \in \{0, 1, rise, fall\}$ is a four valued variable, we only care about the *rise* or $fall$ states at the primary outputs.

The **Justify** routine is used to check if the delay t_d at the primary output node *o* for a state *s* is actually sensitizable. On success, the **Justify** routine returns the set of input

[1]Primary inputs are assigned a level 0, and other nodes are assigned a level which is one larger than the maximum level among all their fanins.

Algorithm 1 Data Dependent Shortest Destabilizing Delay Algorithm

```
Data_Dependent_Short_Destabilize(η)
η* = decompose_and_map(η)
A = levelize(η*)
i = 1
while i ≤ size(A) do
    n = array_fetch(A,i)
    generate_transition_lists (η*, n)
    generate_stable_condition (η*, n)
end while
while 1 do
    (o, s, t_d) = next_shortest_delay(η*)
    vec = Justify (o, s, t_d)
    if vec != NULL then
        break
    end if
end while
```

vector transitions *vec*, which sensitize the delay t_d. If no vector transitions are able to sensitize the delay t_d at the primary output o, then **Justify** returns an empty *vec* and the **next_shortest_delay** routine is called again. The delay corresponding to this **Justify** call is the minimum destabilizing delay for the given circuit η^*. The details of the **Justify** routine are explained in the next section.

III-D. Justify

Justify is a key algorithm in our approach. This algorithm is used to check if a transition (rising/falling) at some time t at a node n in the circuit can be sensitized by one or more primary input vector transitions using a data dependent gate delay model.

As discussed in Chapter II, the sensitizablility of a transition to a state s at time t for a node n is expressed in terms of its transition function $\tau^t_{n,s}$. Further, the transition function at the output of any gate can be expressed in terms of the transition functions of its fanins using a data dependent gate delay model (as explained in the Equations 2.1 and 2.2 in Chapter II). This recursive formulation allows us to express the transition function for any node of the circuit in terms of the transition functions of the primary inputs of the circuit. Now, since the transition function (transitioning to a state s for the node n at time t) is represented only in terms of primary input transitions, we can check for *compatibility* in the primary input vector transitions to determine the sensitizability of the required transition. Consider any *term* of a transition function τ. It contains the logical AND of two other transition functions τ' and τ''. Each of τ' and τ'' are expressed as a set of cubes represented in terms of circuit primary inputs. Each literal of a cube is four-valued, with values $\{0, 1, \text{rise}, \text{fall}\}$. To check if τ' and τ'' is *compatible* we perform a pairwise AND of the cubes of τ' and τ''. If any resulting cube is non-null, $\tau' \cdot \tau''$ is compatible.

Algorithm 2 describes the **Justify** routine. This routine performs two steps: (i) it

Algorithm 2 Justify for node n, checking state s at time t

```
Justify(n, s, t)
if n == primary input then
   return node_state(n, s, t)
end if
for each term ∈ tran_fn(n, s) do
   for each f = fanin(n) do
      s_f = state(term, f)
      d = delay(term , f)
      if s_f = rise/fall then
         if type(d) == range then
            feasible_vec[term][f] = Justify_range(f, s_f, t - min(d), t - max(d))
         else
            feasible_vec[term][f] = Justify(f, s_f, t - (d))
         end if
      else
         if s_f = 0 then
            s_in = fall
            s_in' = rise
         else
            s_in = rise
            s_in' = fall
         end if
         feasible_vec[term][f] = Σ_{0<t_i<t−min(d)} (Justify (f, s_in, t_i) · !Justify_range(f, s_in', t_i, t -
         max(d)))
      end if
   end for
end for
return Compatible(feasible_vec)
```

uses a recursive technique to express the transition function at any node in terms of the transition functions of its immediate fanins and (ii) checks for *conflicts* in the transition function (represented in terms of the primary inputs of the circuit).

The routine is called with node n, transition state s (rising/falling) and a time t. It returns a set of vector transitions at the primary inputs of the circuit, which sensitize the given transition at node n at time t. The first step in this routine is to check if the node is a primary input. For a primary input, the vector transition can be trivially determined by using the checking the arrival time (Equation 2.3. For any other node, the transition

function at the node is evaluated based on the transition functions of it immediate fanins, as explained in Equations 2.1 and 2.2 in Chapter II. The routine computes each *term* of the transition function of the node n at a state s and a time t in a recursive manner in order to finally create the desired transition function at n in terms of transition functions of the immediate fanins of n. Note that the transition function at any node n has a fixed number of *terms*, as evident from the Equations 2.1 and 2.2 of Chapter II.

The **state** routine returns the state s_f of the fanin f of node n for each $term$. The **delay** routine is used to find the delay of the fanin transition. The delay d for each *term* of the transition function of n is different and is also specific to each fanin f. Note that depending on the *term* of the transition function, the delay d may be a range or a single value.

For a rising/ falling state s_f for the fanin f, if d is a range then the **Justify_range** routine is called, else the **Justify** routine is called recursively. The **Justify_range** routine as seen in Algorithm 3 in turn makes successive calls to the *Justify* routine for all possible transition times in the interval t_{min} to t_{max}.

However, if the fanin node state s_f is a stable 1(0) , then the **Justify** routine accumulates all the transition functions which cause the fanin f to rise (fall) before t-d and not fall (rise) till time t. To evaluate this, the **Justify** routine is invoked for all possible rising (falling) times (t_i) between 0 and t - d at node f for the state s_{in}. Also, the transition functions which allow the fanin to fall (rise) between t_i and t are obtained by calling **Justify_range** for the interval t_i to t at the node f for the state $s_{in}^{'}$. Now, by combining the result from the **Justify** call with the compliment of the result from the **Justify_range** call, the final result for the fanin node f to be stable 1(0) in the range $t-d$ to t is obtained. Note that as shown in Figure III.2, to check for an intermediate node n in the circuit to be stable 1 for the time duration t' to t''', we need to generate the transition functions which cause the node n to rise before time t' and not fall until time t'''. Therefore, we need to validate the condition that the same vector transition at the primary inputs which causes n to rise at any

time before t' does not cause it to fall at time t'', where $t' < t'' < t'''$. As a result, we need to maintain the complete list of all possible rise/fall times at a node. Considering a subset of the possible rise/fall transition times may lead to inaccurate results. Thus, the rise/fall arrival time lists cannot be pruned.

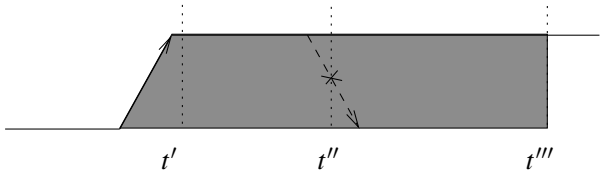


Fig. III.2. Condition for a node to be stable 1 in the time interval t' to t'''

After generating the transition functions for all the fanins of n, the **Compatible** routine checks for the compatibility of the input vector transitions (i.e. it checks for the non-emptiness of the set of transition functions on the primary inputs which would allow the state s to occur at the node n at time t). If the result of the **Compatible** routine results in a non-empty set of vector transitions at the primary inputs, then the node n is sensitizable to a state s at a time t.

III-E. Data Dependent Longest Sensitizable Delay Algorithm

We can easily find the longest sensitizable delay by a small variation in the **Data Dependent Shortest Destabilizing Delay Algorithm**. The only change required is to replace the **next_shortest_delay** routine in the initial algorithm with a **next_longest_delay** routine. This is shown in Algorithm 4. The **next_longest_delay** routine starts by selecting the longest rising or falling delay at any primary output of the circuit. It returns longest a delay t_d (from all the output rise / fall lists created during **generate_transition_lists**), the primary

Algorithm 3 Justify for node n, checking state s during the interval (t_{min},t_{max})

```
Justify_range(n, s, t_min, t_max)
vec = NULL
tran_list = transition_time_list (n, s)
for each t ∈ tran_list do
   if t is in (t_min,t_max) then
      vec += Justify(n, s, t)
   end if
end for
return vec
```

output node o at which the delay may be achieved and the output state s corresponding to the delay t_d. Successive calls to this routine return the next lower value of the delay and the corresponding node and transition values. This is the only change required in the algorithm and all the other steps remain the same.

As explained in Section III-D, we use a recursive formulation to represent the transition function at the output of a node in terms of the fanins of the node. This allows us to represent the transition function at any node in terms of the transition functions at the primary inputs. Using this technique, we start from a primary output and recursively create its transition function, until the transition function is expressed in terms of the primary inputs of the network. Thus, this approach traverses the circuit in a depth first search (DFS) manner. Another possibility is to start from the primary inputs of the network and create the transition functions of all the nodes, in a forward pass of the network (in a breadth first search (BFS) fashion). However, this approach requires the computation of the transition functions for any node at all times and for all states (rise/fall) before the transition function at its immediate fanouts are evaluated. The BFS technique was evaluated and found to be

Algorithm 4 Data Dependent Longest Sensitizable Delay Algorithm

```
Data_Dependent_Long_Sensitize(η)
η* = decompose_network(η)
A = levelize(η*)
i = 1
while i ≤ size(A) do
   n = array_fetch(A,i)
   generate_transition_lists (η*, n)
   generate_stable_condition (η*, n)
end while
while (1) do
   (o, s, t_d) = next_longest_delay(η*)
   vec = Justify (o, s, t_d)
   if vec != NULL then
      break
   end if
end while
```

computationally much more intensive as compared to the DFS technique, since most of the transition functions calculated at a node were never used in the evaluation of the transition function at the node's fanouts. However, it is not known apriori if any such transition function can be pruned before computing the output transition functions. This causes a significant memory utilization, yielding an inefficient approach.

III-F. Implementation

The algorithms above illustrate the use of the recursive technique in computing the true delay of a circuit under the data dependent gate delay model. In small and moderate circuits it is generally easy to construct the transition function and determine the precise path for the minimum destabilizing delay. However, in large circuits the construction of transition functions for each delay may become very intensive. Therefore, some implementation techniques can be utilized to avoid the problem. In this section we will discuss 2 such techniques.

III-F.1. Reduced Search Justify

At each step during the recursive calls in **Justify**, we compute the set of feasible input vector transitions that would allow the final transition to be sensitized. The **Compatible** routine combines the vector transitions generated at the fanins of a node n to determine the sensitizablilty of the transition at n. The compatibility check is required to be done to ensure that the immediate fanins of n do not require conflicting states on the same primary input pin. The condition when the two fanins of a node require conflicting states at any primary input pin is defined as an *incompatibility*.

Let n be any node in the circuit with immediate fanins a and b. The $fanin\_pi_n$ for the node n is defined as the set of primary inputs pins, such that for each primary input

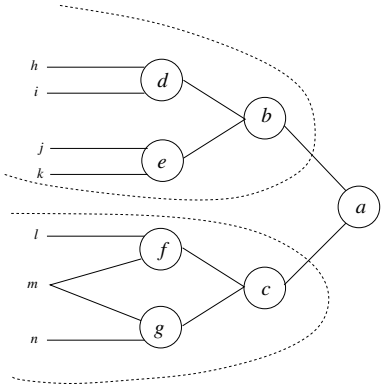


Fig. III.3. Generic representation of a network

$x \in fanin\_pi_n$, there exists a topological path from x to n in the circuit. In the **Justify** routine the primary input vectors transitions returned from the two fanins a and b may have *incompatibility* only if $fanin\_pi_a \cap fanin\_pi_b \neq \emptyset$. This is illustrated in the Figure III.3. For the node p any primary input vector transitions returned by q and r will never have any *incompatibility*. However, at node r the vector transitions returned by f and g may have incompatibility due to the common primary input pin m.

This analysis suggests that if the **Justify** call at any node n (with fanins a and b) returns the primary input vector transitions restricted to the subspace $fanin\_pi_a \cap fanin\_pi_b$, we can still check for compatibility without any compromise on accuracy. This technique of reducing the range of vector transitions returned by **Justify** routine is called **Reduced Search Justify**.

III-F.2. Caching

As discussed above, the **Reduced Search Justify** technique is recursive in nature. A call $\tau^t_{c,fall}$ may be made a large number of times in execution of the top-level algorithm. This

suggests that it may be beneficial to cache the results of **Reduced_Search_Justify** calls. When a call to **Reduced Search Justify** routine with a four-tuple (node *n*, state *s*, time *t* and *fanin_pi*) is made for the first time, the results of the call are stored in the cache. Any subsequent calls with the same set of parameters need not be evaluated again, and the result will be read directly from the cache. To control the memory utilization, we limit the number of entries in the cache. A *least recently used* (LRU) scheme is used to select the entries to be removed, if the total number of entries of the cache increases beyond a fixed number. This number for the total size of the cache is user defined and can be used as a method to control to memory utilization of the whole process.

With the above stated techniques we can enhance the operation of the **Justify** routine without any effect on the accuracy of the results obtained.

CHAPTER IV

EXPERIMENTAL RESULTS

IV-A. Setup

The data dependent sensitizable timing analysis approach was implemented in the logic synthesis environment SIS [17]. The code consists of reading a circuit and mapping it into 2-input NAND gates and inverters. Then the data dependent sensitizable timing analysis (as discussed in Chapter III) is used to analyze the true delay of a circuit. Results are presented for both minimum destabilizing and maximum sensitizable delay analysis. A set of benchmark circuits were used to analyze the effectiveness of our approach.

The gates in the library were characterized for delay in SPICE [16], using a 100nm BPTM [18] process technology. The gate delays for the 2-input NAND gate are given in Table II.1. The rising and falling delays for the inverter are 11.67 ps and 11.63 ps respectively.

IV-B. Destabilizing Minimum Delay Results

The results for the data dependent minimum destabilizing delay are presented in Table IV.1. In this table, Column 1 lists the circuit under consideration. Columns 2 and 3 show the number of primary inputs and primary outputs of the circuit under consideration. Column 4 reports the minimum number of levels of among all the primary outputs of the circuit. The minimum number of levels of a circuit indicates the shortest topological path between the primary inputs and primary outputs for the circuit. Column 5 reports the delay reported by minimum STA (minSTA) . Column 6 reports the minimum destabilizing delay evaluated by our approach. The Column 7 reports the percentage improvement of our results over the results reported by minSTA. The last column reports the total runtime of our approach.

Table IV.1. Comparison of our Minimum Destabilizing Delay approach with minSTA

Ckt.	PI	PO	Levels	minSTA	Our Approach	% Impr.	Time (sec)
C432′	36	3	8	184.56	204.56	10.84	4.55
C499′	41	32	2	73.2	93.2	27.32	25.22
C880′	60	2	4	150.2	190.2	26.63	9.78
C1355′	41	32	2	73.2	93.2	27.32	50.15
C1908′	33	1	4	127	174.67	37.54	60.23
C2670′	233	6	5	114.76	155.24	35.27	11.91
C3540′	50	10	7	169.13	-	-	-
C5315′	178	6	6	199.59	245.39	22.95	87.35
i1′	25	13	2	23.3	23.3	0.00	0
i2′	201	1	6	157.46	181.26	15.11	17.37
i3′	132	4	9	222.89	222.89	0.00	0
i4′	192	4	2	42.13	42.13	0.00	0
i5′	133	66	2	42.13	54.37	29.05	0.1
i6′	138	38	4	119.13	149.5	25.49	1.14
i7′	199	66	4	115.33	141.63	22.80	1.47
i8′	133	17	6	161.26	201.94	25.23	22.3
i9′	88	63	4	119.13	151.37	27.06	23.7
vda′	17	21	9	215.63	259.5	20.35	1.72
table5′	17	13	8	184.56	229.04	24.10	2.19
table3′	14	13	6	172.93	205.13	18.62	3.11
apex1′	45	19	6	172.89	213.57	23.53	2.22
apex3′	54	14	6	173.5	213.5	23.05	1.98
apex4′	9	14	7	215.06	283.14	31.66	4.79
k2′	45	28	7	185.13	227.26	22.76	1.74
rd73′	7	2	6	161.26	213.64	32.48	0.15
alu2′	10	2	10	253.96	317.76	25.12	52.76
duke2′	22	19	5	95.89	115.89	20.86	0.32
clip′	9	3	6	161.26	209.54	29.94	1.3
vg2′	25	3	5	130.8	173.3	32.49	0.4
e64′	65	37	5	126.39	166.91	32.06	0.2
Avg.						24.62	

The results show an increase in the minimum destabilizing delay by 24% on average and 37% in best case, compared to minSTA. In other words minSTA under reports the minimum destabilizing delay of the circuit by 24% on average.

Note for most of these circuits, the shortest topological path is small (typically 1 or 2 levels). To show the effectiveness of our technique, we have removed these short paths wherever possible. In other words Column 4 represents the minimum number of levels after removing the short paths. For C3540 our approach did not finish within a reasonable amount of time (expt. terminated after 1000 secs).

A comparison of the minimum delay paths generated by minSTA and our approach

Table IV.2. Comparison of paths generated by minSTA and our approach

Circuit	Path length for min delay		Identical Path	# of gates with diff. delay	Comments
	minSTA	Our Approach			
C432′	8	8	Yes	1	1 of 4 paths is same
C499′	2	3	No	-	
C880′	4	5	No	-	
C1355′	2	3	No	-	
C1908′	4	6	No	-	
C2670′	5	5	Yes	5	minSTA output rises, our approach output falls
C5315′	6	6	Yes	2	1 of 2 paths is same

is presented in Table IV.2. In this table, Column 1 lists the circuit under consideration. Columns 2 and 3 report the length of the minimum delay path evaluated by minSTA and our approach respectively. Column 4 indicates whether minSTA and our approach have evaluated identical paths. Column 5 lists the number of gates on the path for which the gate delay used by minSTA and our approach was different. Note that the values in this column are shown only for those circuits for which minSTA and our approach obtain identical shortest delay path. In Column 5, comments based on the circuit are given.

The results show that for 4 out of 7 circuits(C499′, C880′, C1355′ and C1908′), minSTA reports different paths as compared to our sensitizable timing analysis approach. In the case of C432′ and C5315′, our approach reports simultaneous transitions on 4 and 2 primary inputs respectively for the minimum destabilizing delay. In both these cases, 1 of the paths is identical to the path evaluated by minSTA. Column 5 reports the number of gates on the identical paths for which minSTA and our approach used different gate delay values. For C2670′, minSTA evaluates a rising delay while our approach evaluates a falling delay for the same path. Thus, the delay values for all the gates in this circuit is different. The above analysis shows that minSTA is not only pessimistic in reporting the minimum delay of the circuit, but also reports an incorrect path (or different transition) in 5 out of 7 test cases. Hence the analysis performed by our approach is more accurate than that of minSTA, validating the utility of our approach.

IV-C. Sensitizable Maximum Delay Results

The results for the data dependent maximum sensitizable delay are presented in Table IV.3. In this table, Column 1 lists the circuit under consideration. Columns 2 and 3 show the number of primary inputs and primary outputs of the circuit under consideration. Column 4 reports the maximum number of levels of among all the primary outputs of the circuit. The levels indicate the longest topological path between the primary inputs and primary outputs of the circuit.

Column 5 reports the delay reported by the STA method. Column 6 reports the delay calculated *sense*, a sensitizable maximum delay analysis tool in SIS [17]. This tool does not account for data dependent delays. The approach of *sense* is reported in [10]. Column 7 reports the maximum sensitizable delay evaluated by our approach. The column 8 and 9 report the percentage improvement of our approach over *sense* and STA respectively. The last column shows the total runtime of our implementation.

The results show a decrease in the maximum sensitizable delay of about 7% on average and about 10% in the best case,compared to *sense*. Note that our approach results in a lower sensitizable delay as compared to both *sense* and STA.

IV-D. Caching

We use caching as a technique to improve the efficiency of our approach. When a cache hit occurs for the **Justify** operation at any node n, it saves the computation time required for all the calls in the transitive fanin of n, which would otherwise be needed to check for the sensitizability of the required transition at n. For this reason, the size of the cache plays an important role in the total runtime. Since the complexity of the **Justify** operation can be estimated by the maximum number of levels in the circuit, we use it as a function to control the cache size. An empirical value c has been defined as a constant (400 in

Table IV.3. Comparison our Maximum Sensitizable Delay approach with Sense and STA

Ckt.	PI	PO	Levels	maxSTA	Sense	Our Approach	% Impr. Sense	% Impr. STA	Run-time (s)
x1	51	35	21	713.14	713.14	684.76	4.14	4.14	1.24
cps	109	24	23	780.27	780.27	726.53	7.40	7.40	2.9
table5	17	15	27	914.21	914.21	863.39	5.89	5.89	131.69
cm150a	21	1	21	798.3	795.3	728.93	9.11	9.52	0.86
cm151a	12	2	18	637.04	634.04	602.16	5.29	5.79	0.24
cmb	16	4	19	736.01	733.01	690.19	6.20	6.64	0.21
clip	9	5	19	651.01	651.01	613.99	6.03	6.03	0.58
vda	17	39	19	646.17	646.17	591.23	9.29	9.29	2.73
apex6	135	99	21	885.32	885.32	823.68	7.48	7.48	1.13
b12	15	9	25	851.92	851.92	805.38	5.78	5.78	0.58
x3	135	99	37	1370.59	1370.59	1257.1	9.03	9.03	43.69
apex3	54	50	25	849.58	849.58	798.92	6.34	6.34	239.7
t481	16	1	31	1048.15	1048.15	975.88	7.41	7.41	9.6
table3	14	14	22	914.21	914.21	855.79	6.83	6.83	29.44
Avg.							6.87	6.97	

our experiments) multiple of the maximum number of levels of the circuit. Figures IV.1 and IV.2 show the variation of runtime based on the size of the cache. These results show that increase in the size of the cache improves the total runtime of our implementation. Further, the degree of improvement decreases with increase in the size of the cache. This is indicated by the flattening of the graph for cache size, greater than $5c$. From Figures IV.1 and IV.2, we note that a cache size greater than $2c$ to $4c$ achieves minimal incremental performance improvements.

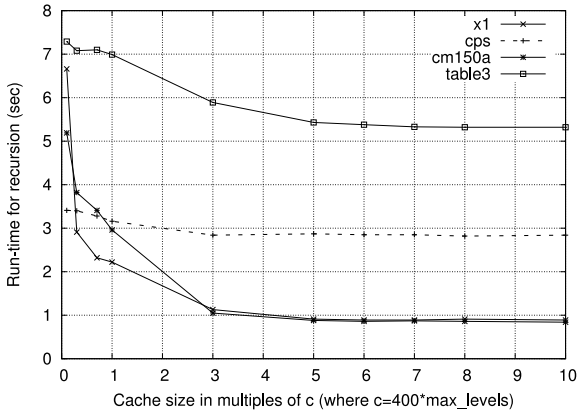


Fig. IV.1. Runtime variation using cache of different sizes

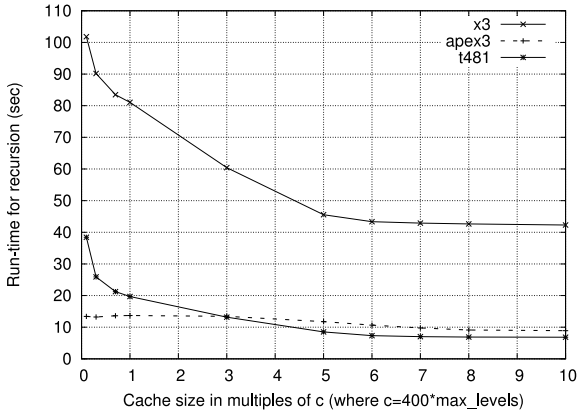


Fig. IV.2. Runtime variation using cache of different sizes-II

CHAPTER V

CONCLUSION AND FUTURE WORK

V-A. Conclusion

Sensitizable timing analysis helps reduce the inaccuracy of topological worst case delay analysis. However, traditional sensitizable timing analysis can be considerably pessimistic due to the pin-to-output gate delay model used. This thesis presents an approach to incorporate "arrival time dependent" delays of a gate in performing sensitizable timing analysis. This approach can be used to evaluate the minimum destabilizing as well as the maximum sensitizable delay of a circuit, using the same formulation. In this approach, we have currently not modeled the gate delay dependence on the input slew and the load capacitance. However, these parameters can be incorporated in the same framework, by utilizing more lookup tables for each gate delay (based on different values of the input slew and the output load capacitance). For high fanout nodes, the dependence of the gate delay on output capacitance may be large. This would result in a smaller change in the gate delay based on different transitions at the gate inputs. Thus, the overall improvement of our approach may be less.

The results of our approach underline the effectiveness of using the data dependent gate delay model in evaluating the minimum destabilizing and maximum sensitizable delay of a circuit. From the results presented in this thesis, we can also say that the effect of data dependence is more pronounced while computing the minimum destabilizing delay of the circuit, and results in an improvement of about 24% as compared to minSTA.

V-B. Future Work

The effectiveness of our method can be improved by using a fully implicit representation for input vector transitions. This would allow us to use efficient logic minimization techniques like *multi-valued ESPRESSO* [19] to represent the input vector transitions. Also MDDs (multi-valued decision diagrams) may be used for an implicit representation of the input vector transitions.

Another technique to improve the efficiency of our approach could be to partition the circuit into slices of maximum depth k, in a topological manner from the primary inputs of the circuit. Each slice would be analyzed independently, with the results of slice i being used as arrival times for the slice i+1. This approach would result in some loss of accuracy, which can be traded off against the size of each slice.

The gate library size can be increased by including higher fanin gates. This will enhance the applicability of the approach presented in this thesis for any general static CMOS circuit.

REFERENCES

[1] S. T. Huang, T. M. Parng, and J. M. Shyu, "A new approach to solving false path problem in timing analysis," in *Proc. of the Intl. Conf. on Computer-Aided Design*, Nov 1991, pp. 216–219.

[2] H. C. Chen and D. H. C. Du, "Path sensitization in critical path problem," *IEEE Transactions on Computer-Aided Design*, pp. 196–201, Feburary 1993.

[3] P. McGeer, A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis and Optimization*, chapter Delay Models and Exact Timing Analysis, pp. 167–189, Kluwer Academic Publishers, 1993.

[4] Y. Kukimoto, W. Gosti, A. Saldanha, and R. K. Brayton, "Approximate timing analysis of combinational circuits under the XBD0 model," in *Proc. of the Intl. Conf. on Computer-Aided Design*, Nov 1997, pp. 176–181.

[5] L.-R. Liu, H.-C. Chen, and D. H.-C. Du, "The calculation of signal stable ranges in combinational circuits," in *Proc. of the Intl. Conf. on Computer-Aided Design*, Nov 1991, pp. 312–315.

[6] R. B. Hitchcock Sr., G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware.," *IBM Journal of Research and Development*, vol. 26, no. 1, pp. 100–105, Jan 1982.

[7] J. Benkoski, E. V. Meersch, L. J. M. Claesen, and H. D. Man, "Timing verification using statically sensitizable paths," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Sept 1990, vol. 9, pp. 1073–1084.

[8] D. Brand and V. S. Iyengar, "Timing analysis using functional analysis," in *IEEE Transactions on Computers*, Oct 1988, vol. 37, pp. 1309–1314.

[9] J. P. M. Silva and K.A. Sakallah, "An analysis of path sensitization criteria," in *Proc. of the IEE International Conference on Computer Design*, Oct 1993, pp. 68–72.

[10] P. C. McGeer and R. K. Brayton, "Efficient algorithms for computing the longest viable path in a combinational network," in *Proc. of the Design Automation Conf.*, New York, NY, USA, 1989, pp. 561–567, ACM Press.

[11] D.H.C. Du, S.H.C. Yen, and S. Ghanta, "On the general false path problem in timing analysis," in *Proc. of the Design Automation Conf.*, New York, NY, USA, June 1989, pp. 555–560, ACM Press.

[12] S.-W. Cheng, H.-C. Chen, D. H.-C. Du, and A. Lim, "The role of long and short paths in circuit performance optimization," in *Proc. of the Design Automation Conf.*, Los Alamitos, CA, USA, 1992, pp. 543–548, IEEE Computer Society Press.

[13] R. P. Llopis, L. R. Xirgo, and J. C. Bordoll, "Short destabilizing paths in timing verification," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computer & Processors*, Washington, DC, USA, 1994, pp. 160–163, IEEE Computer Society.

[14] S.-Z. Sun, D. H.-C. Du, and H.-C. Chen, "Efficient timing analysis for CMOS circuits considering data dependent delays," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computer & Processors*, Washington, DC, USA, 1994, pp. 156–159, IEEE Computer Society.

[15] L.-C. Chen, S. K. Gupta, and M. A. Breuer, "A new gate delay model for simultaneous switching and its applications," in *Proc. of the Design Automation Conf.*, New York, NY, USA, 2001, pp. 289–294, ACM Press.

[16] L. Nagel, "SPICE: A computer program to simulate computer circuits," Tech. Rep. UCB/ERL M520, Univ. of California, Berkeley, CA, May 1995.

[17] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA, May 1992.

[18] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proc. of IEEE Custom Integrated Circuit Conference*, Jun 2000, pp. 201–204, Available at http://www-device.eecs.berkeley.edu/ ptm.

[19] R. Rudell and A. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for multiple-valued logic minimization," in *Proceedings of the Custom International Circuit Conference (CICC-85)*, May 1985, pp. 230–234.

VITA

Karandeep Singh received his Bachelor of Engineering degree in Electronics and Electrical Communication Engineering from the Punjab Engineering College, Chandigarh, India in June 2002. . He has worked for ST Microelectronics India, Noida, India. In August 2005, he joined Texas A&M University to pursue his master's degree in computer engineering. His research at Texas A&M is focused on improvement of timing analysis techniques for static CMOS based digital circuit design. He received his M.S. in Computer Engineering in May 2007.

Permanent Address:
Karandeep Singh
331-A WERC,
Texas A&M University,
College Station, TX-77843
E-mail: karandeep@tamu.edu

The typist for this thesis was Karandeep Singh.